

Mono vers stéréo

Projet final, IFT6145 - Vision tridimensionnelle

Sylvain Laporte - Automne 2020

Plan de match

- Pourquoi mono vers stéréo ?
- Présentation de l'article "Learning Stereo from Single Images"
 - Le problème
 - Leur solution
 - Leurs résultats
- Notre projet: implémenter et explorer les idées de l'article
 - Outils
 - Méthode
 - Résultats et difficultés
- Pour aller plus loin

Pourquoi mono vers stéréo ?

Pourquoi mono vers stéréo ?

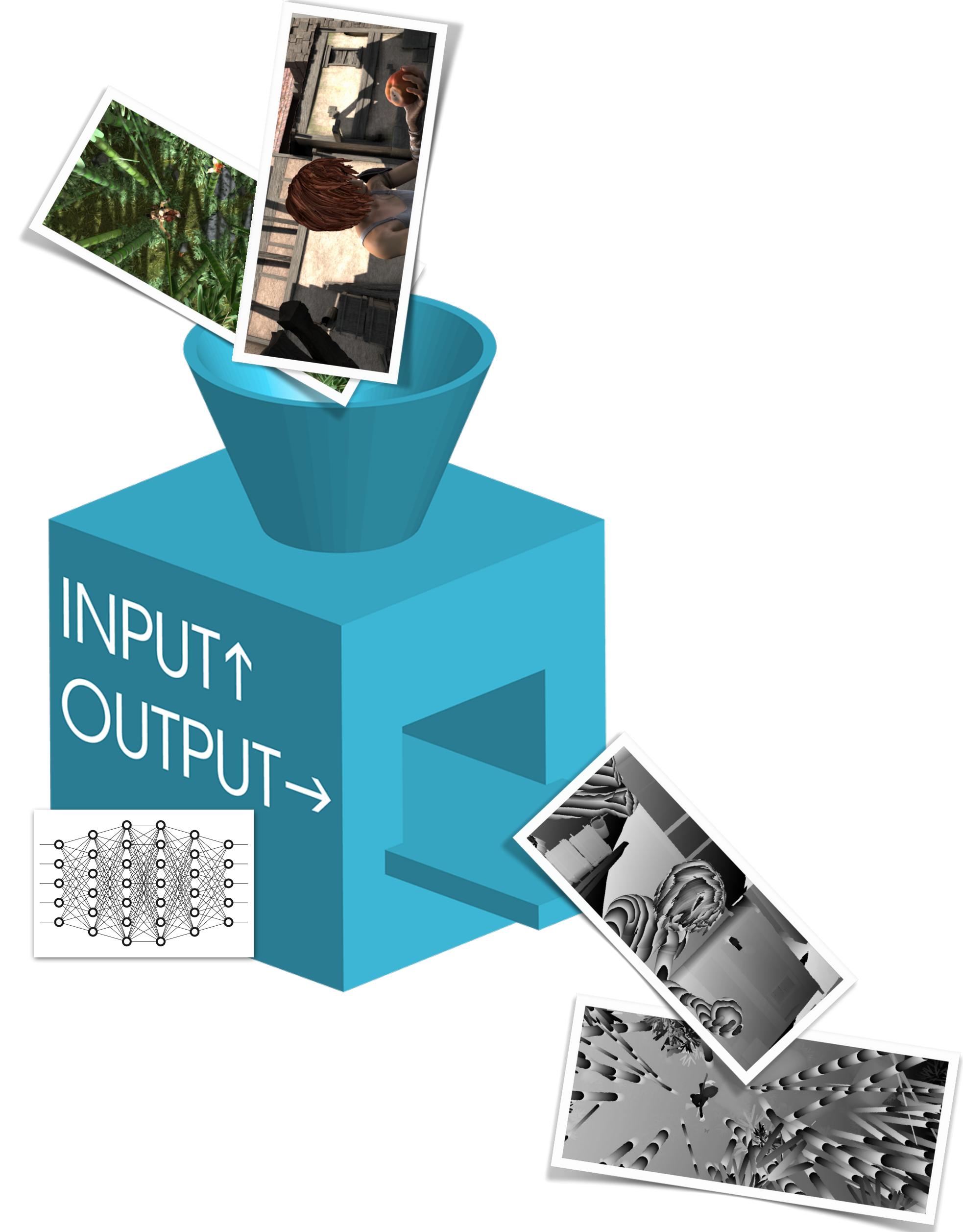
Intérêt pour l'industrie du cinéma

- Selon la VES, l'industrie du cinéma met présentement plus l'accent sur la post-conversion 2D-3D comme technique pour la sortie de films en stéréo que sur la sténographie native, en déclin.
- La vaste majorité des productions stéréo sont typiquement capturées en mono, puis converties en stéréo en post-production.
 - Moins coûteux, peut être donné en sous-traitance.
 - Les productions récentes ont investies en ce sens en recherche et innovation.

Objectif de notre projet

Exploration

- Explorer les techniques de vision permettant de produire, à partir d'une seule image, des cartes de disparités.
- Y mettre un peu de *deep learning*.
- Point de départ : l'article « Learning stereo from single images ».



L'article

L'article

« Learning Stereo From Single Images »

Learning Stereo from Single Images

Jamie Watson¹ Oisin Mac Aodha² Daniyar Turmukhambetov¹
Gabriel J. Brostow^{1,3} Michael Firman¹

¹Niantic ²University of Edinburgh ³UCL

Abstract. Supervised deep networks are among the best methods for finding correspondences in stereo image pairs. Like all supervised approaches, these networks require ground truth data during training. However, collecting large quantities of accurate dense correspondence data is very challenging. We propose that it is unnecessary to have such a high reliance on ground truth depths or even corresponding stereo pairs. Inspired by recent progress in monocular depth estimation, we generate plausible disparity maps from single images. In turn, we use those flawed disparity maps in a carefully designed pipeline to generate stereo training pairs. Training in this manner makes it possible to convert any collection of single RGB images into stereo training data. This results in a significant reduction in human effort, with no need to collect real depths or to hand-design synthetic data. We can consequently train a stereo matching network from scratch on datasets like COCO, which were previously hard to exploit for stereo. Through exten-

Le problème

Un détour... *raccourci !*

- Comment générer plus facilement des données d'entraînement pour des réseaux profonds de correspondance stéréo ?
- Les meilleures méthodes stéréo restent à apprentissage supervisé, lesquelles requièrent de larges *datasets*.
- Acquérir ces données est laborieux et cher.



Travaux précédents

Des données difficiles à acquérir

- Pour des scènes réelles, acquérir ces données demande des senseurs spécifiquement conçu pour détecter la profondeur, en plus des systèmes de caméras pour capturer les images.
- Les senseurs de profondeur, souvent basés sur le laser, limitent le types de scènes que l'on peut capturer, ex. : pas d'éclairage sous soleil direct.

Travaux précédents

Réutiliser des données existantes est difficile

- D'autres sources de données existent :
 - des vidéos sur le Web
 - des films 3D
 - des images multi-vues
- Le problème est que ces données viennent souvent sans les informations sur la correspondance (*ground truth*).
- Estimer la disparité sur ses images est particulièrement difficile à cause de problèmes tels que des disparités négatives (films 3D) et des changements d'apparence extrêmes dans le temps (images multi-vues).

Travaux précédents

Synthétiser des données

- Il est possible d'entraîner les réseaux sur des données recueillies de scènes 3D générées par ordinateur.
- Plus facile à générer et contrôle complet des propriétés des caméras virtuelles.
- Toutefois, les meilleures performances des réseaux sont obtenues lorsque les données se rapprochent des images originales, et la plupart des réseaux seront affinés sur des données réelles.
- Aussi : augmenter les données naturelles avec des données synthétiques.

La solution

Générer une paire stéréo à partir d'une seule image

- Prendre une image d'une caméra conventionnelle (mono) et générer la seconde vue stéréo à partir de la première.
- Avantages :
 - les images de base sont triviales à obtenir;
 - ces images correspondent à la scène, contrairement aux données synthétiques;
 - dans la mesure où ces paires serviront ultimement à entraîner un réseau de correspondance stéréo, leur approximation suffit.

Méthode

D'une seule image à une paire d'images stéréo

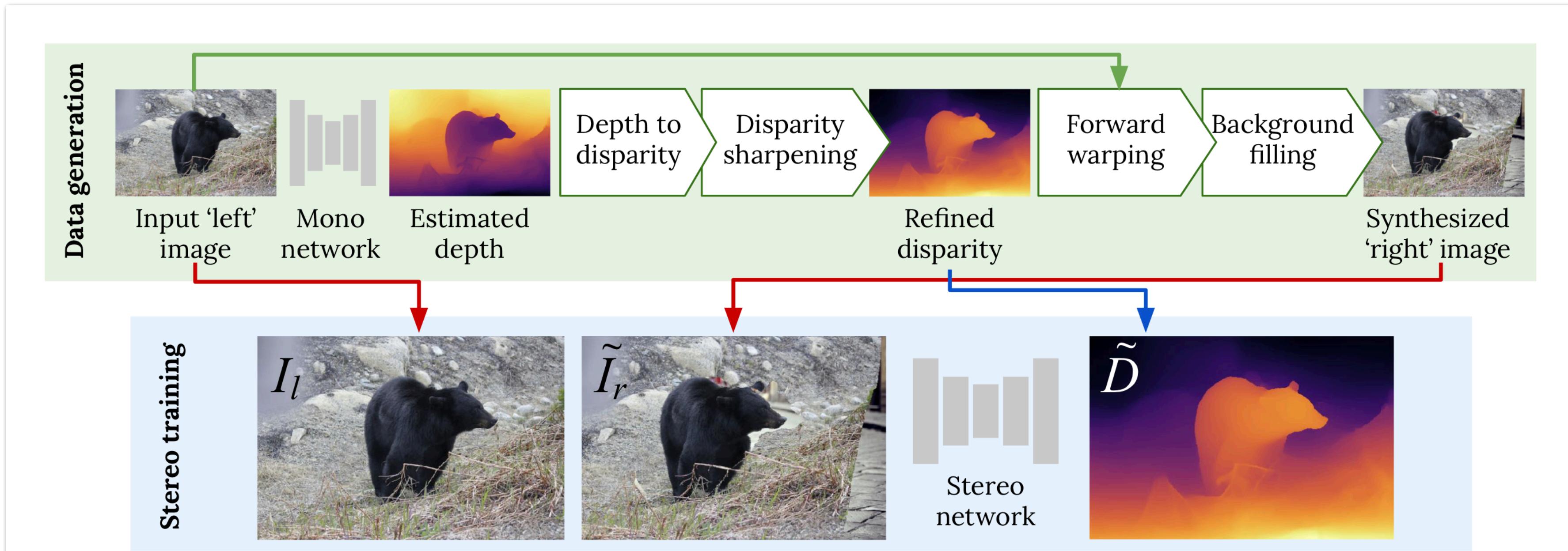
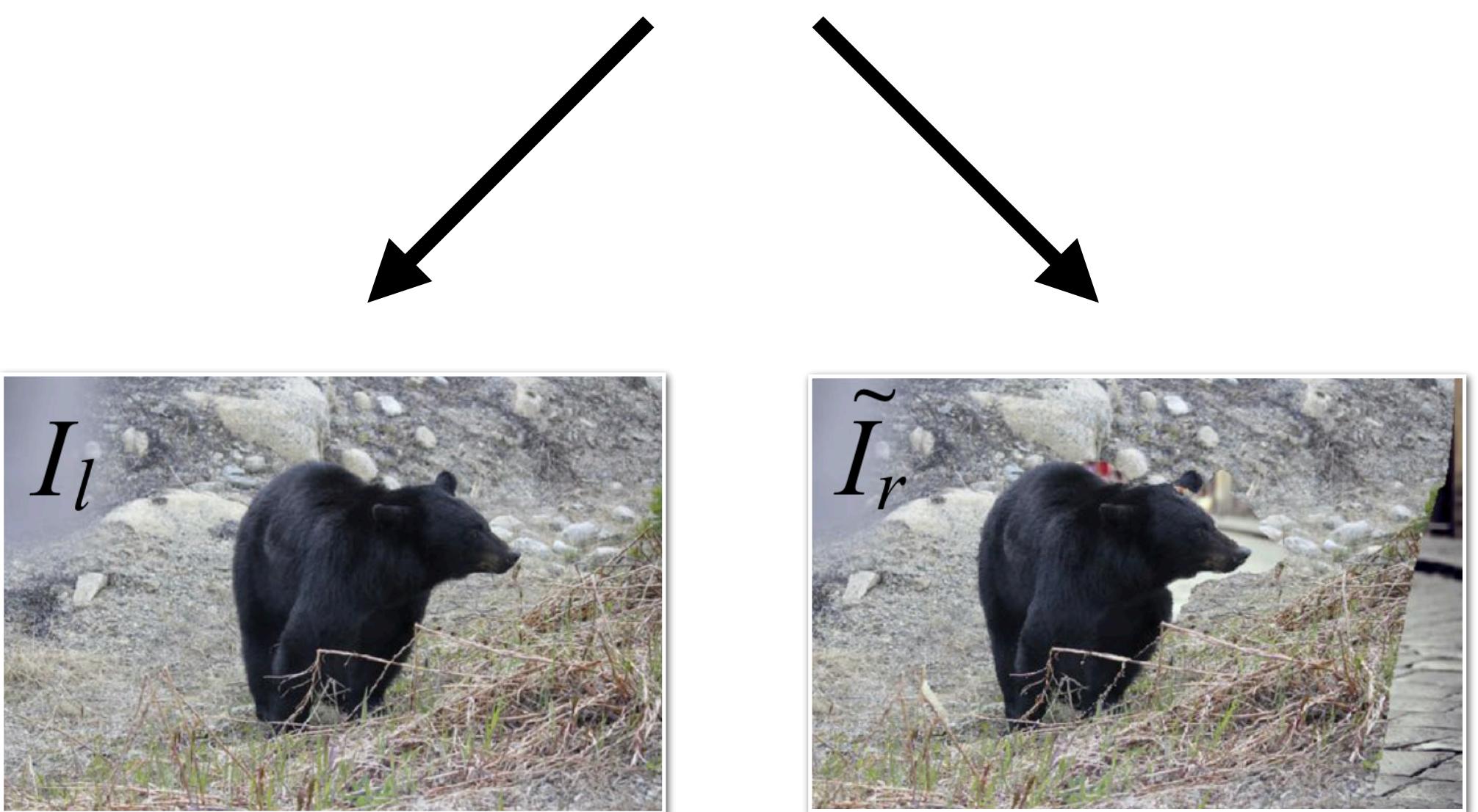


Fig. 2: Overview of our data generation approach. We use off-the-shelf monocular depth estimation to synthesize stereo training pairs from single images. We convert estimated depth to a sharpened disparity map, and then forward warp pixels from the input left image I_l to create a plausible and geometrically accurate right image \tilde{I}_r .

Méthode

Les étapes

1. Estimer un *depth map* à l'aide d'un réseau mono.
2. Raffiner le *disparity map* :
 1. *Depth sharpening*.
3. Transformer ce *depth map* en *disparity map*.
4. Générer la seconde image :
 1. *Forward warping*.
 2. Corriger les occlusions.
 3. Remplissage de l'arrière-plan.



Méthode

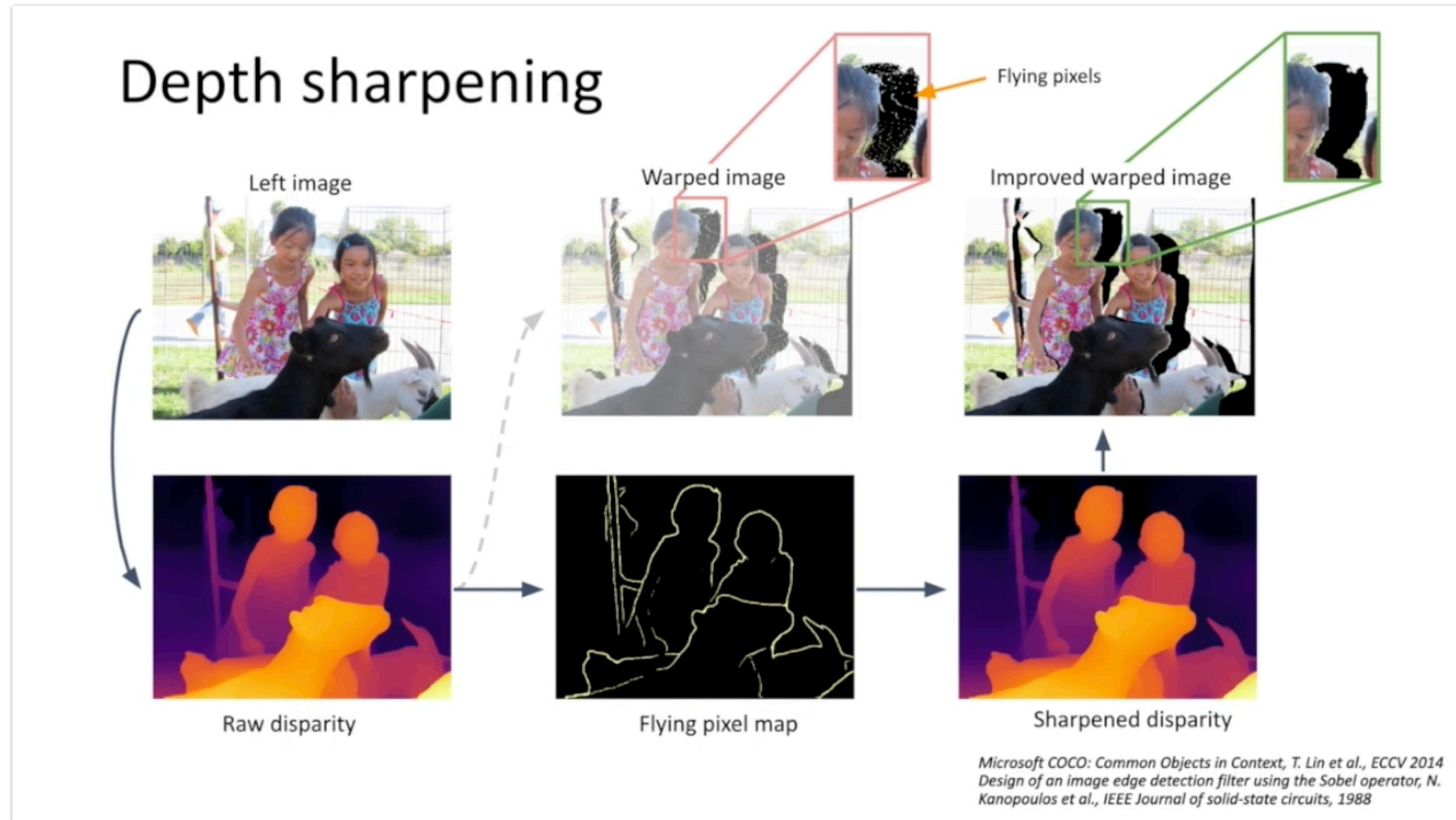
1. Estimer un *depth map* à l'aide d'un réseau préentraîné

- Exemples de réseaux :
 - MiDaS
 - DiW
 - Monodepth2
 - Megadepth



Méthode

3. Raffiner le *depth map* - *Depth sharpening*



Méthode

2. Transformer le *depth map* en *disparity map*

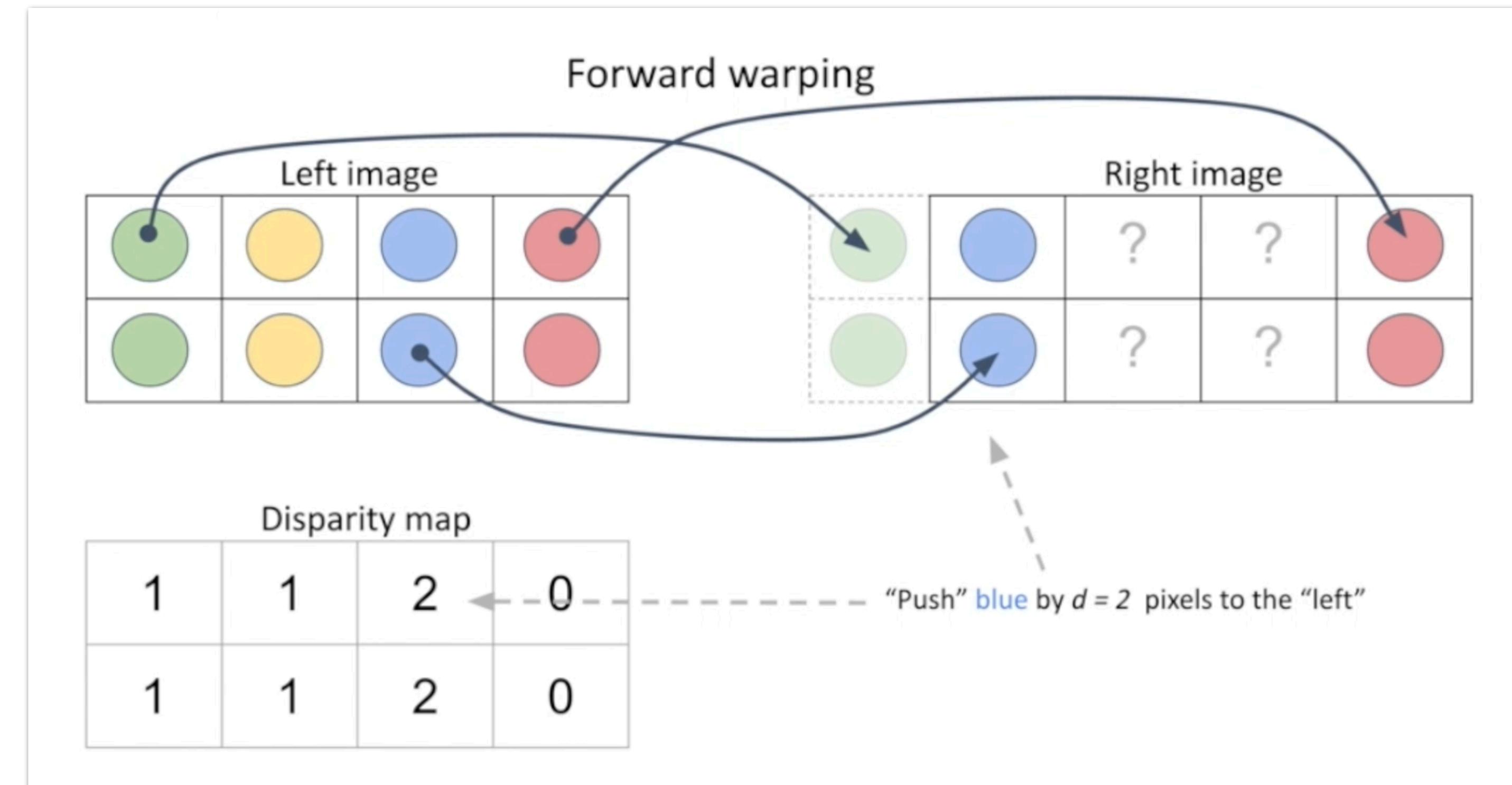
$$\tilde{D} = \frac{sZ_{max}}{Z}$$

où :

- s est un facteur d'échelle échantillonné aléatoirement (uniformément entre $[d_{min}, d_{max}]$)
- Z est la *depth map*
- Z_{max} est la profondeur maximale

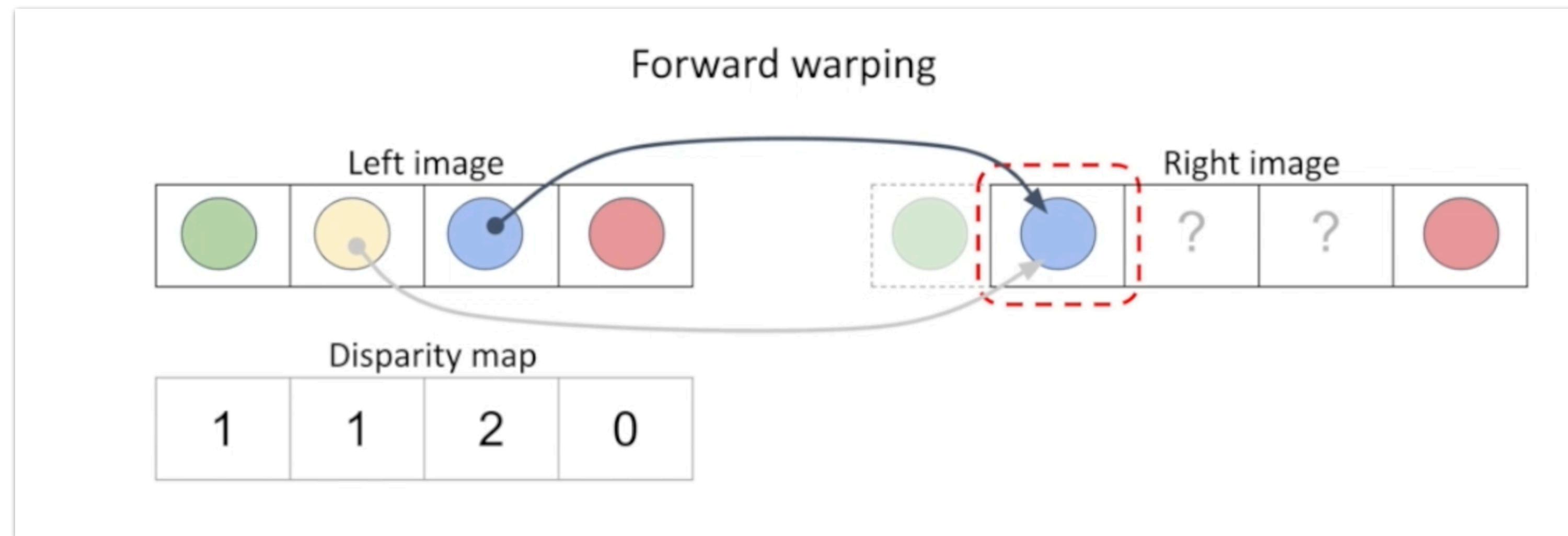
Méthode

4. Générer la seconde image par *forward warping*



Méthode

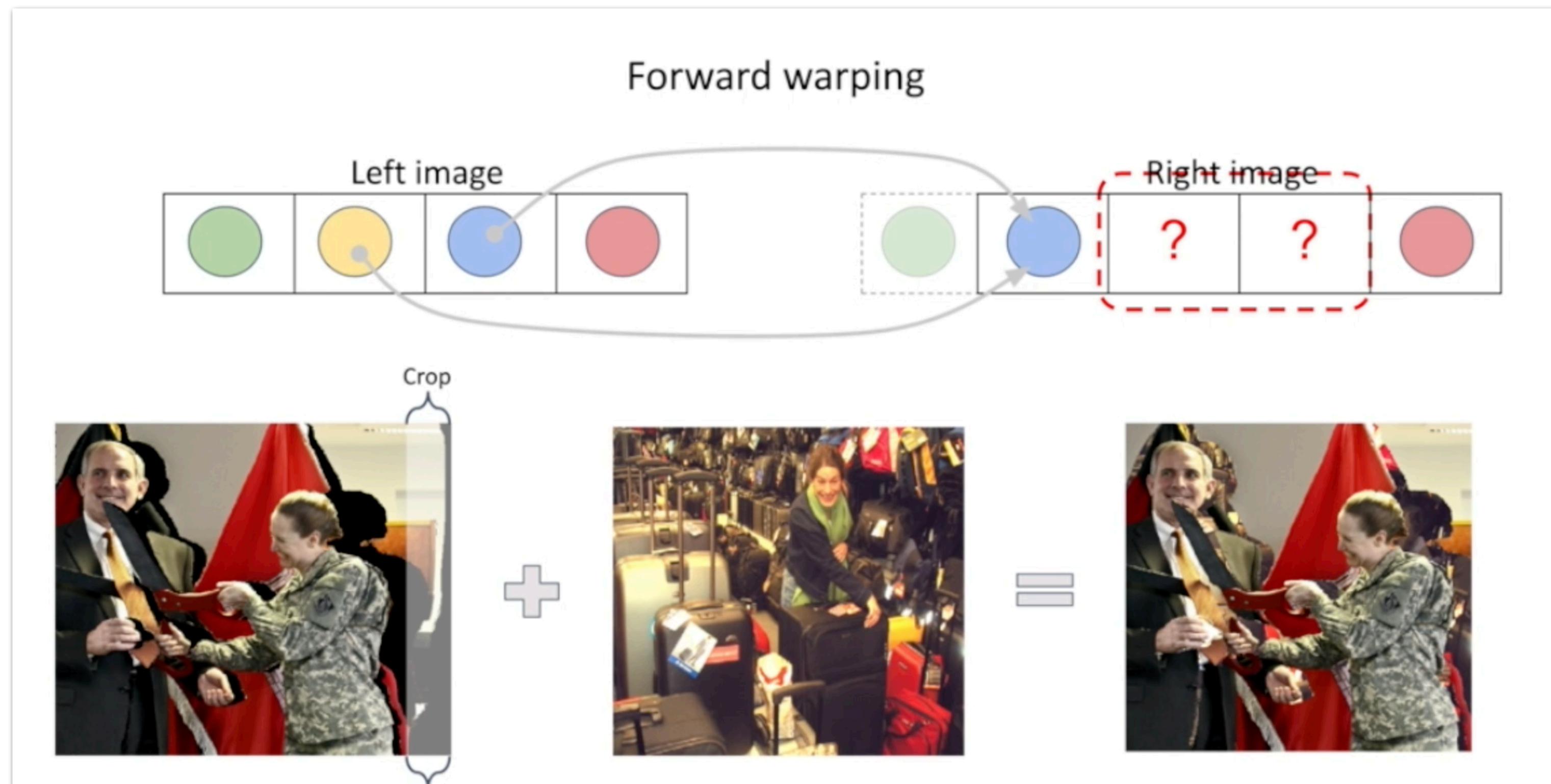
3. Générer la seconde image - Corriger les occlusions



On choisit le pixel ayant la plus grande disparité (donc le plus près de la caméra).

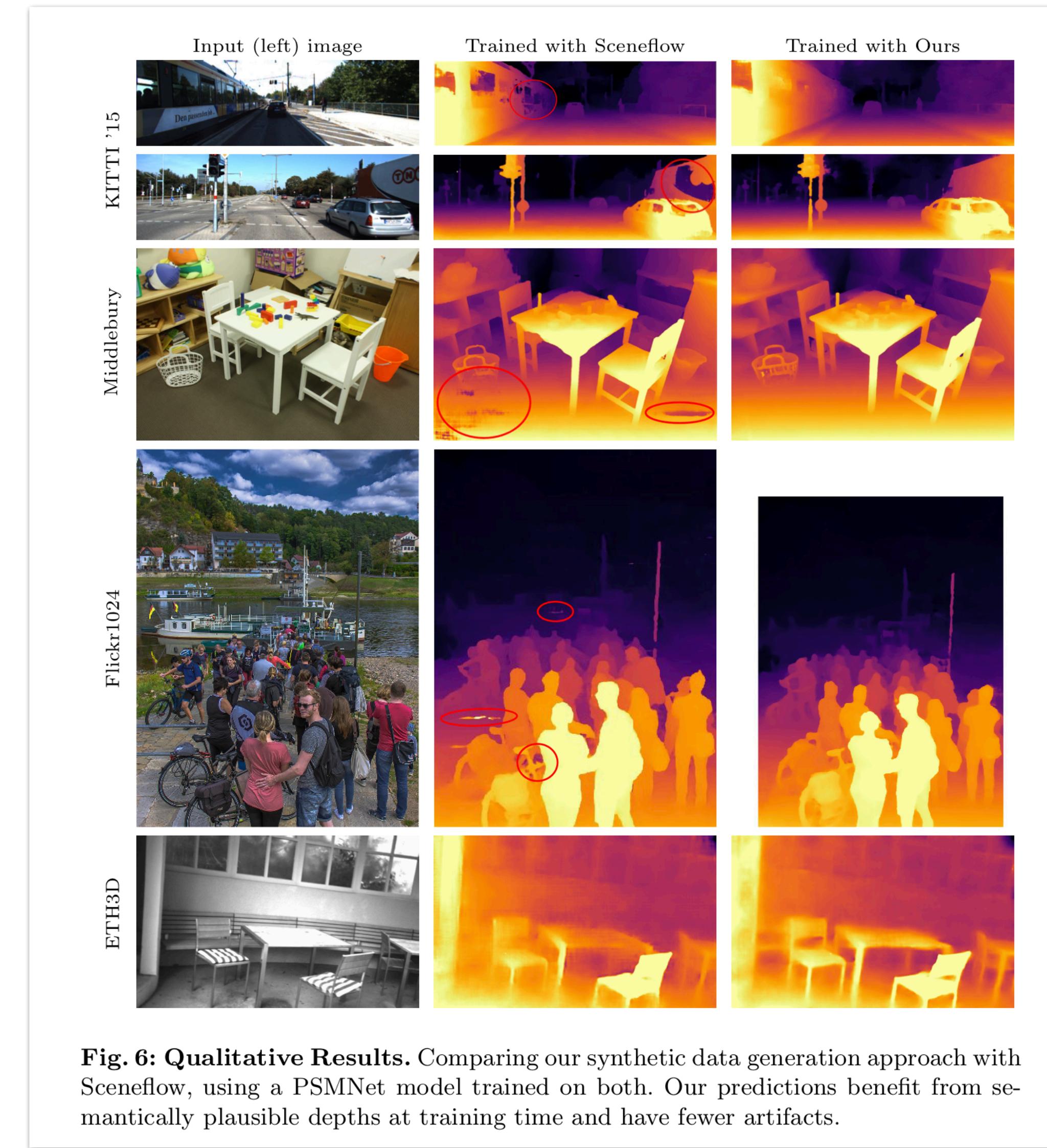
Méthode

4. Générer la seconde image - Remplir l'arrière-plan



On recadre et on superpose l'image générée à une image de tonalité semblable.

Leurs résultats



Leurs résultats

Différences entre les *disparity maps* estimé mono et prédit par un réseau stéréo



Fig. 4: We can recover from errors in monocular depth estimation. Problems present in monocular depths (b), such as missing objects and uneven ground do not transfer to our eventual stereo predictions (c).

Leurs résultats

Comparatif des méthodes de génération de données d'entraînement

Table 1: Our approach outperforms alternative stereo training data generation methods. Each row represents a single PSMNet trained with a different data synthesis approach, without any dataset-specific finetuning. We can see that simple synthesis methods still produce valuable training data. However, our approach in the bottom row, which incorporates a monocular depth network [50], outperforms even synthetic Sceneflow data. Numbers here are directly comparable to Table 3, showing we beat the baselines no matter what depth network or training data we use.

Synthesis approach	Training data	KITTI '12 EPE >3px	KITTI '15 EPE >3px	Middlebury EPE >2px	ETH3D EPE >1px
Affine warps (e.g. [12])	MfS	3.74 14.78	2.33 14.94	21.50 61.19	1.28 24.21
Random pasted shapes (e.g. [43])	MfS	2.45 11.89	1.49 7.57	11.38 40.37	0.77 14.19
Random superpixels	MfS	1.28 6.08	1.23 5.90	12.02 32.92	2.01 13.95
Synthetic	Sceneflow [44]	1.04 5.76	1.19 5.80	9.42 34.99	1.25 13.04
SVSM selection module [41]	MfS	1.04 5.66	1.10 5.47	8.45 31.34	0.57 11.61
Ours	MfS	0.95 4.43	1.04 4.75	6.33 26.42	0.53 8.17

Leurs résultats

La méthode fonctionne indépendamment au type de réseau stéréo

Table 2: Our approach is agnostic to stereo model architecture. We compare three diverse stereo architectures, from the lightweight iResnet [39], through to the computationally expensive, but state-of-the-art, GANet [79]. Rows marked with † use the publicly released model weights. We see that our synthesized data (MfS) consistently performs better than training with the synthetic Sceneflow dataset.

Architecture	Training data	KITTI '12 EPE >3px	KITTI '15 EPE >3px	Middlebury EPE >2px	ETH3D EPE >1px
iResnet [39]	Sceneflow	1.07 6.40	1.15 5.98	12.13 41.65	1.57 17.90
iResnet [39]	MfS	0.93 5.38	1.15 5.75	8.52 33.36	0.61 11.65
PSMNet [5]†	Sceneflow	6.06 25.38	6.04 25.07	17.54 54.60	1.21 19.52
PSMNet [5]	Sceneflow	1.04 5.76	1.19 5.80	9.42 34.99	1.25 13.04
PSMNet [5]	MfS	0.95 4.43	1.04 4.75	6.33 26.42	0.53 8.17
GANet [79]†	Sceneflow	1.30 7.97	1.51 9.52	11.88 37.34	0.48 8.19
GANet [79]	Sceneflow	0.96 5.24	1.14 5.43	9.81 32.20	0.48 9.45
GANet [79]	MfS	0.81 4.31	1.02 4.56	5.66 24.41	0.43 6.62

Notre projet : implémenter et
explorer les idées de l'article

Intentions

Partons la mer est belle...

- A. Implémenter la méthode de l'article pour générer des paires stéréo d'images pour l'entraînement de réseaux profonds de correspondance stéréo.
- B. Tester nos paires en entraînant un réseau de correspondance stéréo (PSMNet).

Les outils

Ce que nous avons utilisé pour coder le projet

- Python
- PyTorch – pour utiliser les réseaux profonds
- Google Colab – parce que Apple 😠 CUDA
- OpenCV et Matplotlib – pour l'I/O des images

Les outils

Données et réseaux existants

- Réseaux existants
 - MiDaS – pour l'estimation de la profondeur mono
 - PSMNet – comme réseau de correspondance stéréo (comme l'article)
- Datasets
 - COCO 2017 – pour les images mono (comme l'article)
 - KITTI 2015 – pour tester sur le réseau stéréo

A. Implémentation

Initialiser MiDaS depuis PyTorch

```
You, seconds ago | 1 author (You)
10 class Mono2Stereo():
11     """Générateur de paire d'images stéréo à partir d'une seule image.
12
13     Utilise le modèle de génération de depth map MiDaS.
14     """
15
16     def __init__(self, use_large_model=True):
17         """Télécharge et initialise le modèle MiDaS sur le GPU.
18
19         Args:
20             use_large_model (bool, optional): Selectionne le modèle large. Par défaut: True.
21         """
22
23         # Télécharger le modèle MiDaS
24         if use_large_model:
25             self.midas = torch.hub.load("intel-isl/MiDaS", "MiDaS")
26         else:
27             self.midas = torch.hub.load("intel-isl/MiDaS", "MiDaS_small")
28
29         # Déplacer le modèle sur le GPU
30         self.device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
31         self.midas.to(self.device)
32
33         # Initialiser le modèle
34         self.midas.eval()
35
36         # Télécharger les transformations incluses avec MiDaS
37         midas_transforms = torch.hub.load("intel-isl/MiDaS", "transforms")
38
39         if use_large_model:
40             self.transform = midas_transforms.default_transform
41         else:
42             self.transform = midas_transforms.small_transform
```

A. Implémentation

Méthode complète pour générer la vue de droite

```
43     def generate_right_view(self, dataset, d_min=0, d_max=192):
44         """Méthode complète pour générer une paire d'images stéréo à partir d'une seule image."""
45
46         results = []
47
48         for image in dataset:
49             # Prétraitement de l'image pour inférence MiDaS
50             # Applique la normalisation MiDaS et déplace l'image vers le GPU
51             input_batch = self.transform(image).to(self.device)
52
53             # Prédit le depth map
54             with torch.no_grad():
55                 prediction = self.midas(input_batch)
56
57                 # Redimensionne l'image à sa taille originale
58                 prediction = torch.nn.functional.interpolate(
59                     prediction.unsqueeze(1),
60                     size=image.shape[:2],
61                     mode="bicubic",
62                     align_corners=False,
63                 ).squeeze()
64
65             Z = prediction.cpu().numpy()
66
67             # Convertit le depth map en disparity map
68             D_tilde = self.__compute_D_tilde(Z, d_min, d_max)
69             D_tilde = np.clip(D_tilde, d_min, d_max)
70
71             # Calcule la vue de droite
72             right_image = self.__compute_right_view(image, D_tilde, d_max)
73
74             # Crop images
75             image = image[:, -d_max:, :]
76             right_image = right_image[:, -d_max:, :]
77             D_tilde = D_tilde[:, -d_max:, :]
78
79             results.append((image, right_image, D_tilde))
80
81         return results
82
```

A. Implémentation

Méthodes auxiliaires - Convertir en *disparity map* et générer la vue de droite

```
82
83     def __compute_D_tilde(self, Z, d_min, d_max):
84         """Convertit un depth map en disparity map"""
85
86         s = np.random.randint(d_min, d_max)    # Randomly sampled scaling factor
87         Z_max = np.max(Z)
88
89         return (s * Z_max) / Z
90
91     def __compute_right_view(self, img, disp_map, d_max):
92         """Calcule la vue de droite à partir d'une image et de sa disparity map"""
93
94         disp_map = disp_map.astype(int)
95         right_image = img.copy()  # Pour interpolation
96
97         for p_i in range(img.shape[0]):
98             for p_j in range(img.shape[1]):
99                 # Forward warp
100                 new_p_j = p_j - disp_map[p_i, p_j]
101                 if new_p_j > 0:
102                     right_image[p_i, new_p_j] = img[p_i, p_j]
103
104                 # Interpolation
105                 right_image[p_i, p_j] = img[p_i, p_j]
106
107         return right_image
```

A. Implémentation

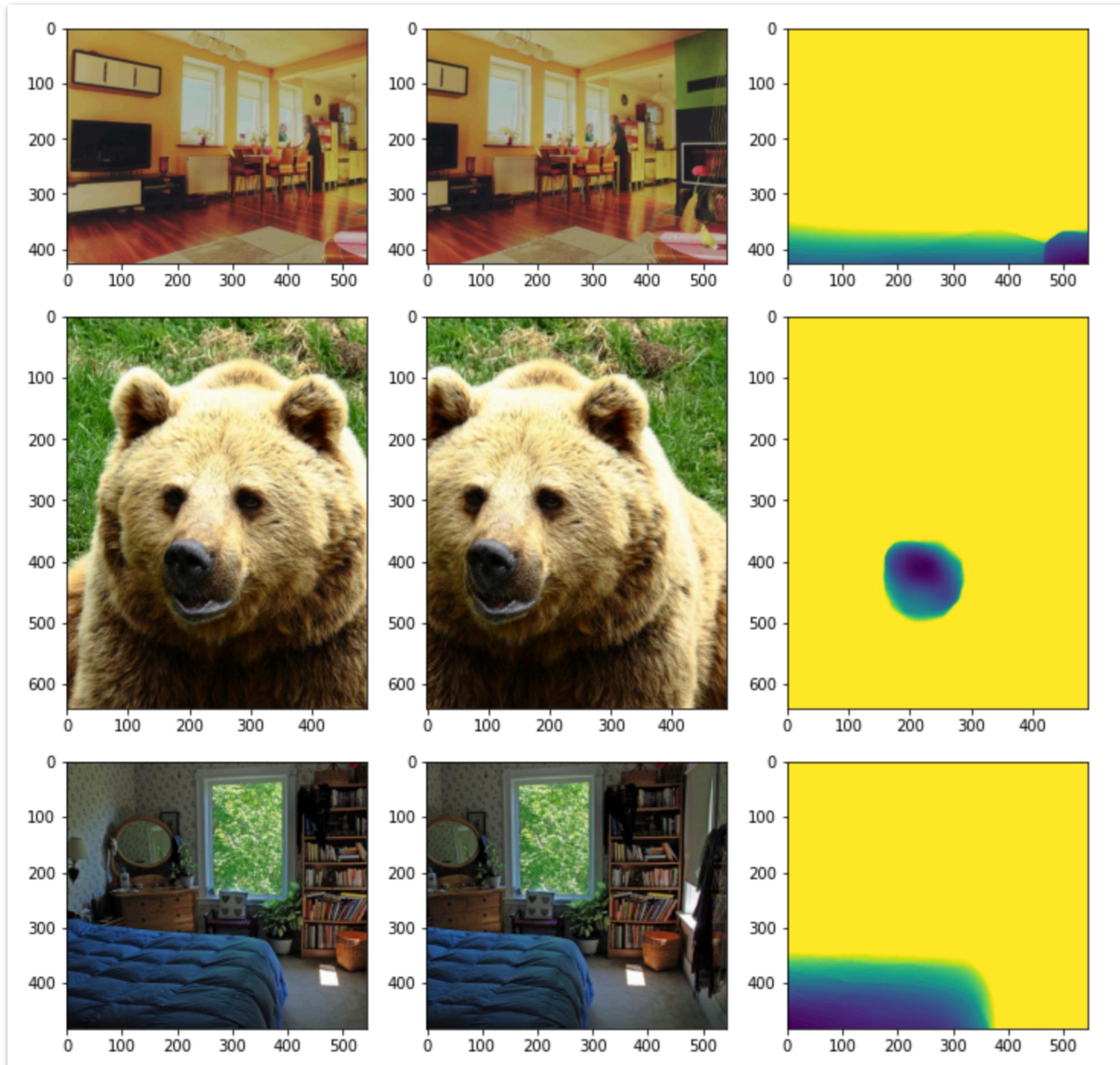
Détails et difficultés

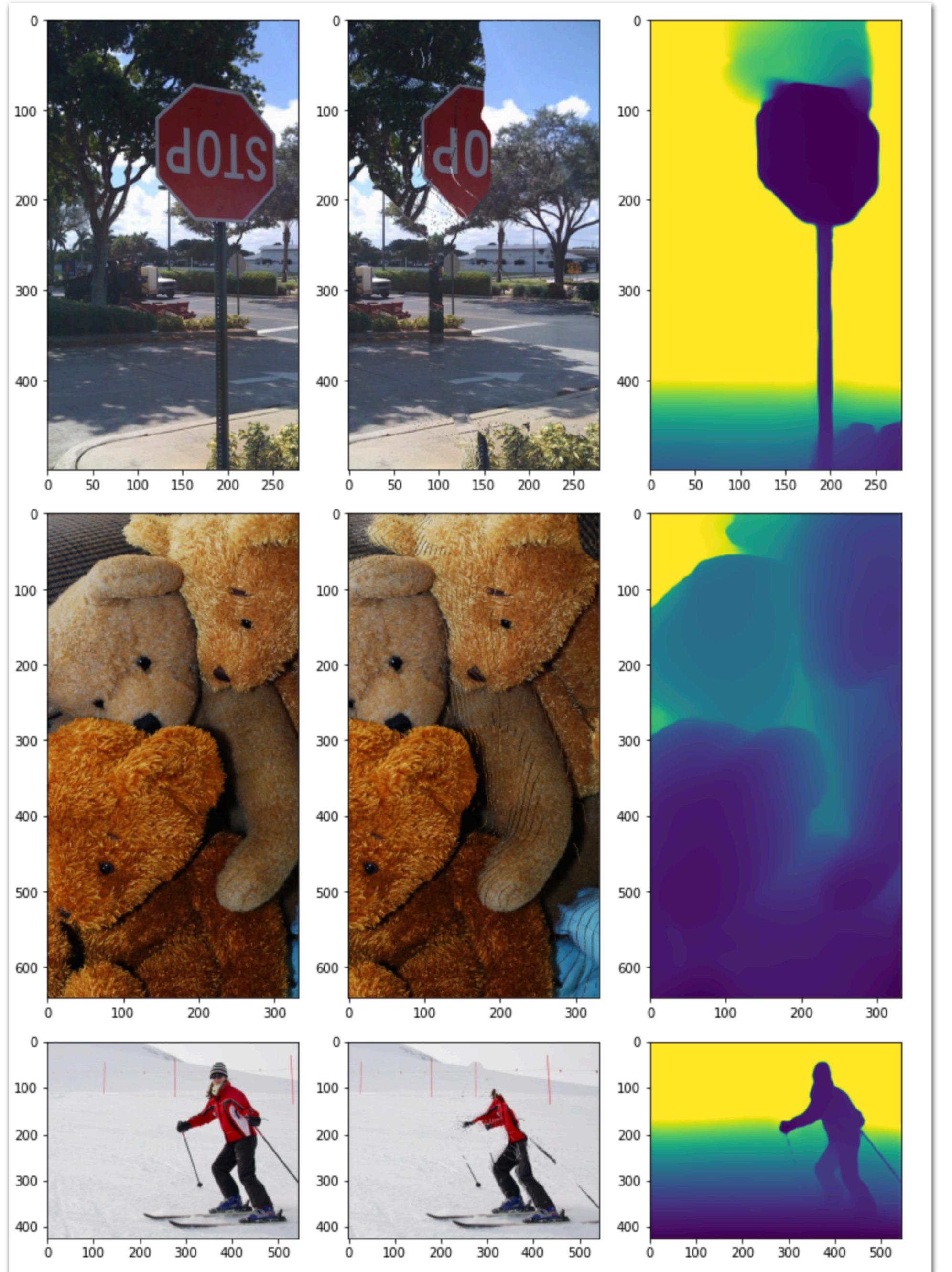


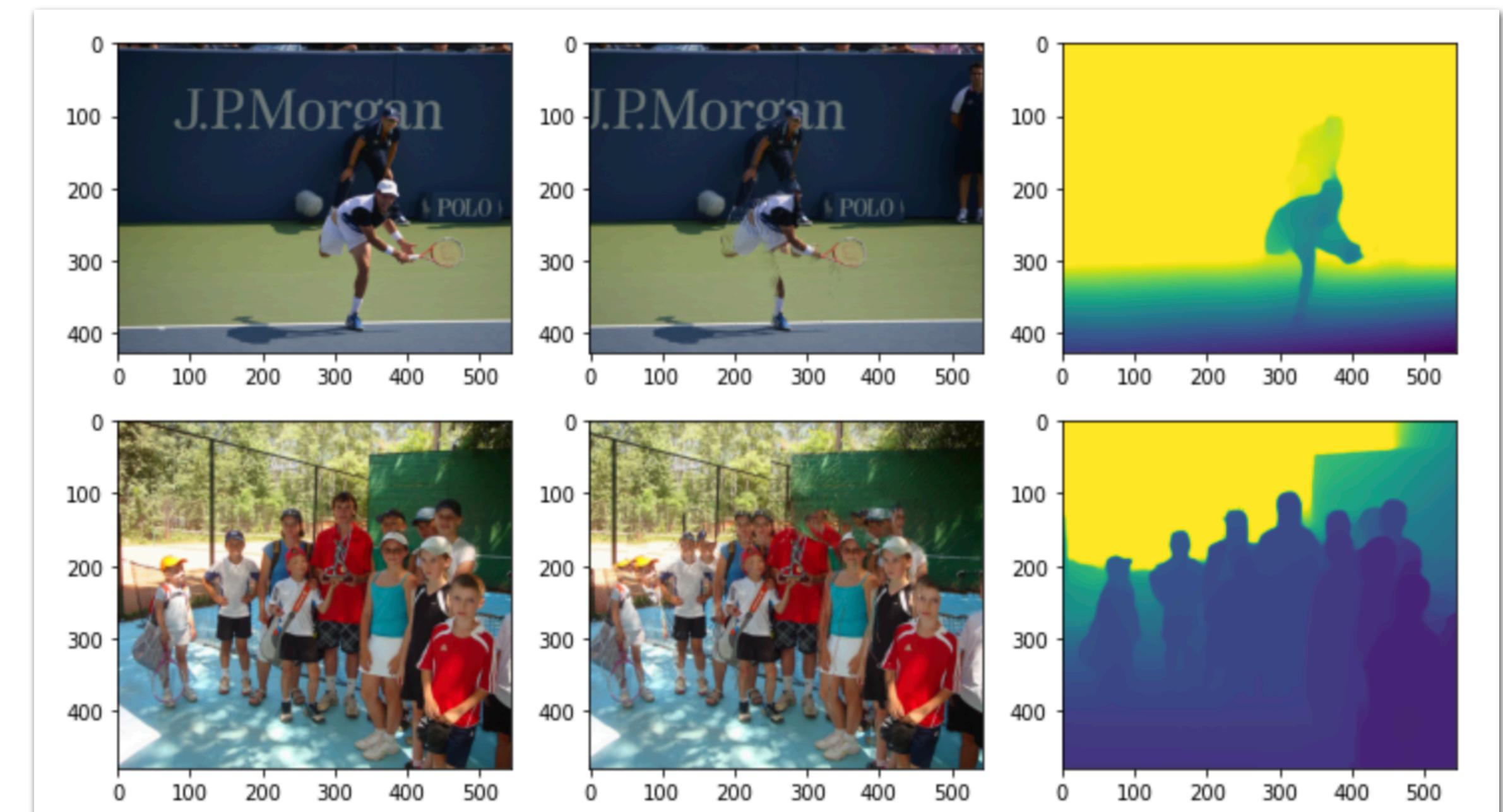
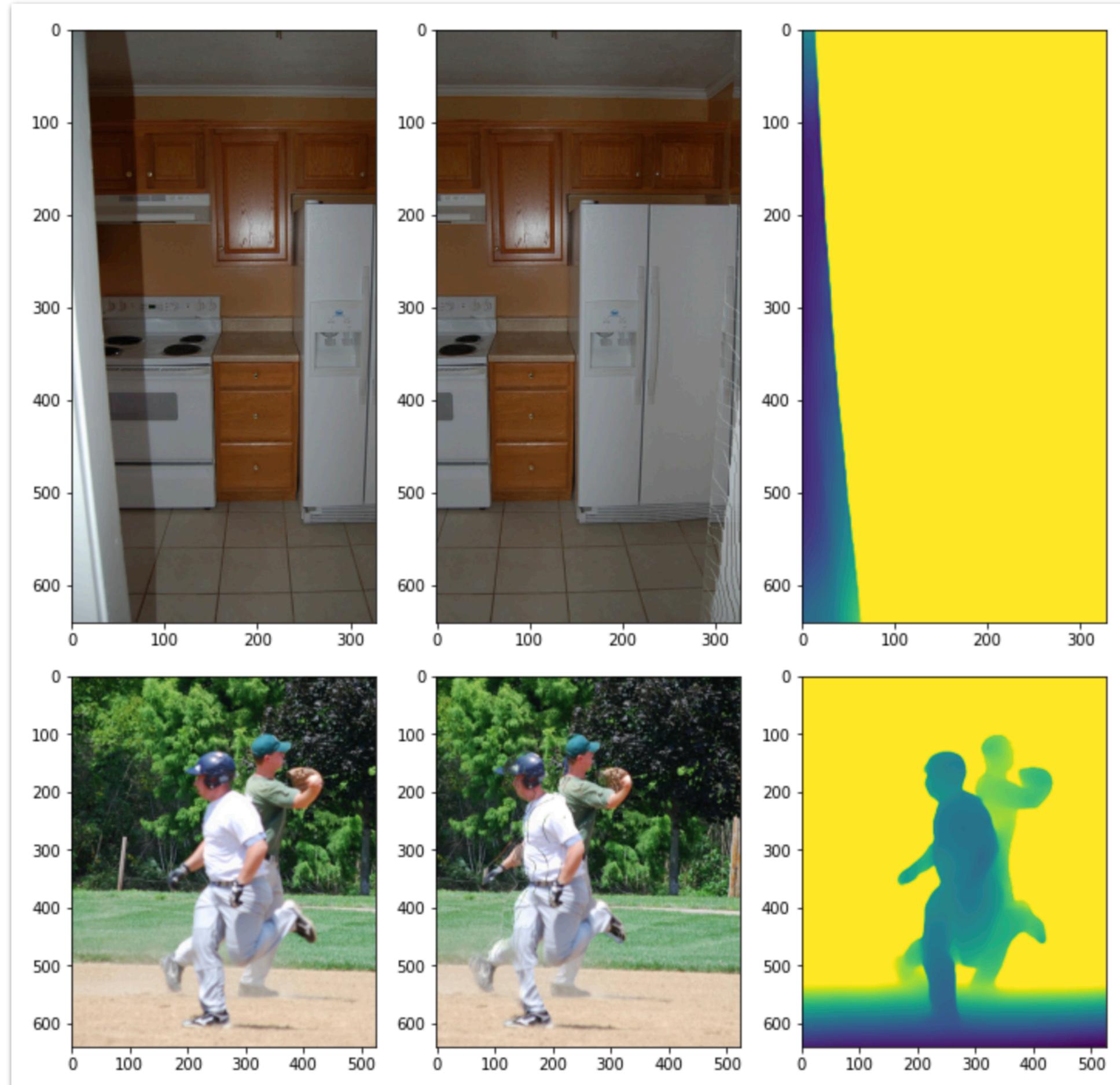
- Pour remplir l'arrière-plan, nous avons réutilisé l'image originale à défaut d'avoir une image du même contexte. Assez proche de la solution de l'article.
- Le *depth sharpening* a posé plus de problèmes : le changement des pixels dont la réponse filtre Sobel dépassait 3 nous a donné de mauvais résultats. À terme, nous avons sauté cette étape.
- Certains résultats semblent très bons, alors que d'autres le sont moins.

A. Implémentation

Résultats







B. Tests

Méthode projetée

- Inférer sur un *dataset* KITTI 2015 à l'aide d'un réseau profond de correspondance stéréo PSMNet pré-entraîné sur SceneFlow, comme dans l'article.
- Entraîner un réseau PSMNet sur un *dataset* COCO 2017, converti en paires stéréo par la méthode implémentée.
- Comparer les deux à l'aide des valeurs :
 - *end-point-error* (EPE)
 - thresholded error rate ($> 3px$)

B. Tests

Valeurs de test

```
5
6     import numpy as np
7     from numpy.linalg import norm
8
9     ∵ def compute_EPE(predicted_disp, ground_truth):
10        |    return np.mean(norm(predicted_disp - ground_truth, ord="fro", axis=(1, 2)))
11
12    ∵ def compute_TER(predicted_disp, ground_truth):
13        |    tau = 3
14
15        |    height, width, depth = predicted_disp.shape
16        |    nb_pixels = height * width * depth
17
18        |    diff = np.abs(predicted_disp - ground_truth)
19        |    return diff[np.where(diff > tau)].size / nb_pixels * 100
```

B. Tests

Problème : le modèle PSMNet

- Nous avons tenté d'utiliser l'implémentation officielle, mais y avons trouvé des bugs :
 - modules non importables
 - Erreur de type (argument de ligne de commande)
 - Mauvaise indentation
- Malgré la correction de ces bugs, des erreurs de taille de tenseurs restaient présentes.

B. Tests

Problème : le modèle PSMNet

- Aussi, en exécutant le script de test fourni pour une image, nous avons obtenu des résultats plutôt différents de ceux promis.



Nos résultats



Ground truth

EPE : 8396424.6677885807 > 3px : 99.98625



B. Tests

« This case will remain unsolved. »

– Conclusion de la majorité des dossiers X-files



Conclusions

Conclusions

- Les paires d'images stéréo générées depuis des images mono semblent permettre d'entraîner des réseaux de correspondance plus performants que les données issues de scènes synthétiques.
- Il serait peut-être possible d'envisager regrouper la méthode de l'article en un algorithme permettant de produire des cartes de disparités convaincantes à partir d'une seule image.

Merci !